Sandia National Laboratories

Exceptional service in the national interest

ME469: Code Walk Through- CVFEM and EBVC

Stefan P. Domino[1,2]

[1] Computational Thermal and Fluid Mechanics, Sandia National Laboratories

[2] Institute for Computational and Mathematical Engineering, Stanford

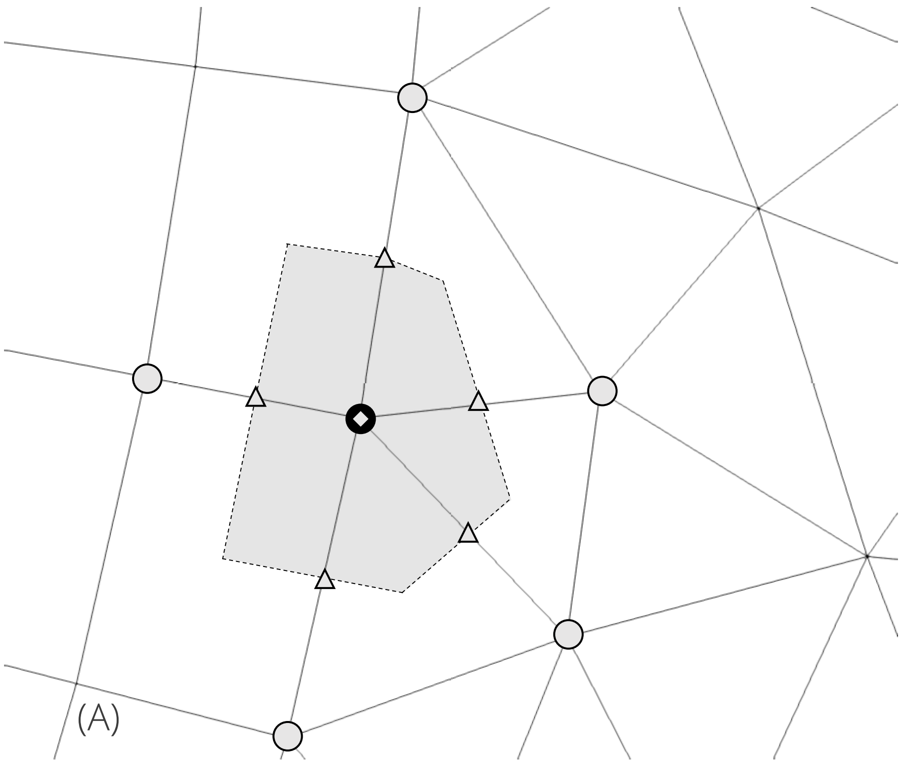
SAND2018-4536 PE

U.S. DEPARTMENT OF ENERGY

NNSA

Edge-Based Vertex-Centered Leverages the Dual-Volume Element-based Description of a Control Volume Finite Element Method (CVFEM)

- EBVC (A) and CVFEM (B) – As shown below, the dual-volume and integration point layout is very similar



Deep Dive on CVFEM

- CVFEM is a discretization scheme that:
  - Iterates over locally-owned elements for Time/Source/etc. (volumetric-based terms)
  - Iterates over locally-owned elements for Advection/Diffusion/etc. (integrated by parts terms)

Below is the patch of elements connected to node 2 (a global matrix row number)

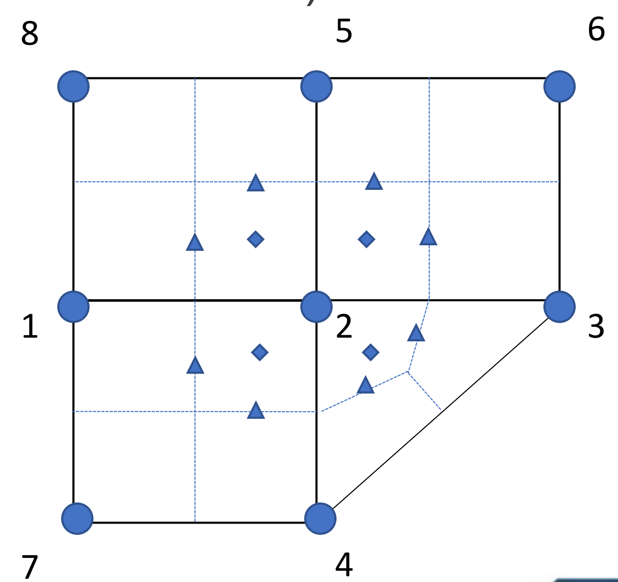
- A *dual-volume* is defined within each element

Interpolation functions, or shape functions: $\phi_{ip} = \sum_n N_n^{ip} \phi_n$

- Volume and Surface examples:

$$\int w \frac{\partial \rho \phi}{\partial t} dV \approx \sum_{ip} \frac{\left(\gamma_1 \rho_{ip}^{n+1} \phi_{ip}^{n+1} + \gamma_2 \rho_{ip}^{n} \phi_{ip}^{n} + \gamma_3 \rho_{ip}^{n-1} \phi_{ip}^{n-1}\right)}{\Delta t} V_{ip}$$

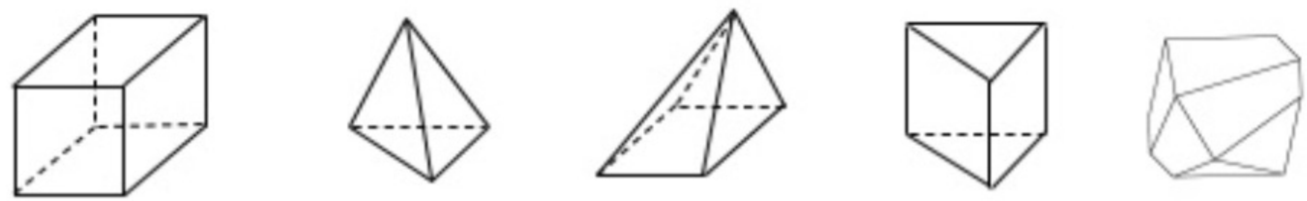
$$\int w \frac{\partial q_j}{\partial x_j} dV \approx -\sum_{ip} \frac{\mu}{Sc}_{ip} \frac{\partial \phi}{\partial x_j}_{ip} n_j dS = -\sum_{ip} \frac{\mu}{Sc}_{ip} \sum_{nd} \frac{\partial N_{nd}^{ip}}{\partial x_j} \phi_{nd} A_j^{ip}$$

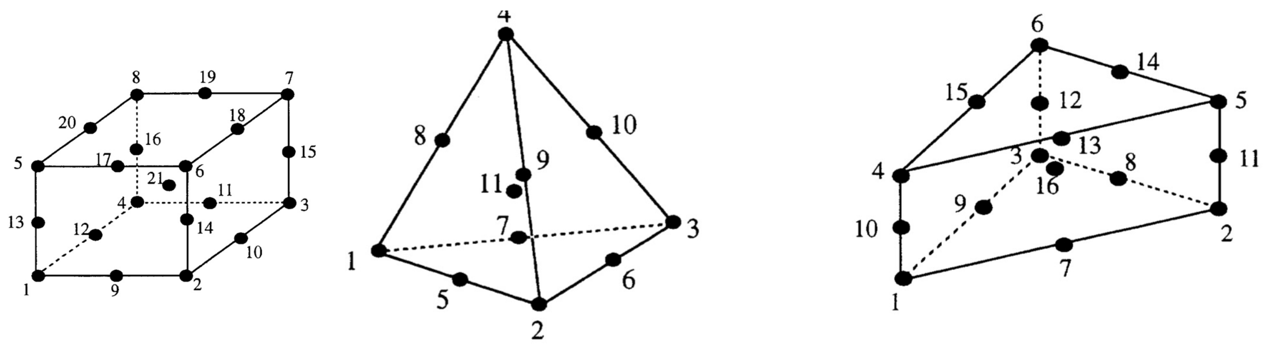


Sample patch of elements

Examples of Various Topologies

Hex8 | Tet4 | Pyramid5 | Wedge6 | Arbitrary





Higher-order promoted elements (Hex27, Tet10, Wedge16, Hex64, etc.)

Code Design: Managing a Hybrid Mesh

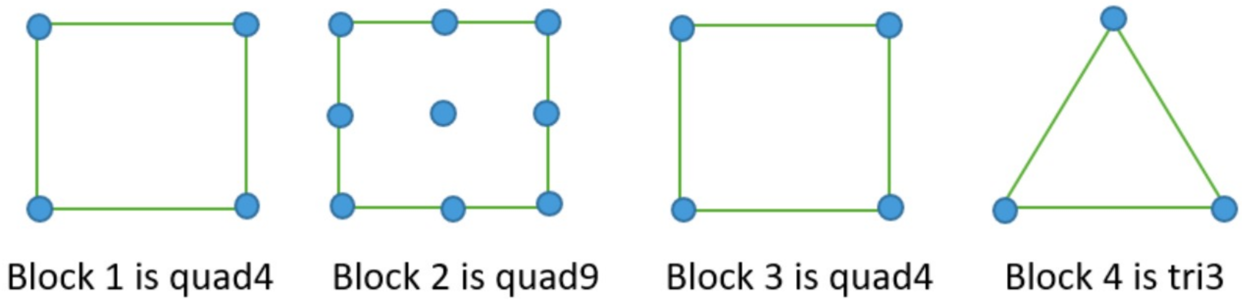


Figure 4: Heterogeneous topologies example.

Attributes:

- Three unique topologies: quad_4, quad9, tri3
- Iterate a set of “parts” that map to the homogeneous element blocks
- Abstract out the integration rule, i.e., AlgTraits::nodesPerElement_, etc.
- Create one algorithm per element topology type (avoids resizing)

Sample Nalu ElemKernel: Construction

```
template<typename AlgTraits>
MomentumNSOElemKernel<AlgTraits>::MomentumNSOElemKernel(
  ElemDataRequests& dataPreReqs)
{
  // define master element rule for this kernel
  MasterElement *meSCS
    = sierra::nalu::MasterElementRepo::get_surface_master_element(AlgTraits::topo_);

  // add ME rule
  dataPreReqs.add_cvfem_surface_me(meSCS);

  // add fields to gather
  dataPreReqs.add_coordinates_field(*coordinates_, AlgTraits::nDim_, CURRENT_COORDINATES);
  dataPreReqs.add_gathered_nodal_field(*velocityNp1_, AlgTraits::nDim_);

  // add ME calls
  dataPreReqs.add_master_element_call(SCS_GIJ, CURRENT_COORDINATES);
}
```

Listing 4: Attributes of a kernel; part A the constructor.

Constructor defines the fields to gather and the element operations required, e.g., dndx, area_vector, etc.

Sample Nalu ElemKernel: Execution

<u>Design Point</u>: *Consolidated* approach hides the element loop!

```
template<typename AlgTraits>
void
MomentumNSOElemKernel<AlgTraits>::execute(
  SharedMemView<DoubleType**>& lhs,
  SharedMemView<DoubleType *>& rhs,
  ScratchViews<DoubleType>& scratchViews)
{
  SharedMemView<DoubleType**>& v_uNp1
    = scratchViews.get_scratch_view_2D(*velocityNp1_);
  SharedMemView<DoubleType***>& v_gijUpper
    = scratchViews.get_me_views(CURRENT_COORDINATES).gijUpper;

  for ( int ip = 0; ip < AlgTraits::numScsIp_; ++ip ) {

    // determine scs values of interest
    for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {

    // assemble each component
    for ( int k = 0; k < AlgTraits::nDim_; ++k ) {

      // determine scs values of interest
      for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {

        // save off velocityUnp1 for component k
        const DoubleType& ukNp1 = v_uNp1(ic,k);

        // denominator for nu as well as terms for "upwind" nu
        for ( int i = 0; i < AlgTraits::nDim_; ++i ) {
          for ( int j = 0; j < AlgTraits::nDim_; ++j ) {
            gUpperMagGradQ += constant*v_gijUpper(ip,i,j);
          }
        }
      }
    }
  }
}
```

Thread-local scratch arrays using A Kokkos SharedMemView

Templated

SIMD

MD-array rather than error-prone pointer arithmetic

Listing 5: Attributes of a kernel; part B the body.

Implicit Solves - Refresher

We wish to solve in *residual* form (also known as delta-form)

Start with standard: $Ax^{k+1} = b$

Transform to residual form: $A\Delta x^{k+1} = b - Ax^k = -res$

Where M need not be equal to A:

$$M\Delta x^{k+1} = b - Ax^k = -res$$

For a given time step:

do while (!converged) {

$$\begin{bmatrix} \frac{\partial}{\partial p}C & \frac{\partial}{\partial \tilde{u}_x}C & \frac{\partial}{\partial \tilde{u}_y}C & \frac{\partial}{\partial \tilde{z}}C \\ \frac{\partial}{\partial p}\tilde{U}_x & \frac{\partial}{\partial \tilde{u}_x}\tilde{U}_x & \frac{\partial}{\partial \tilde{u}_y}\tilde{U}_x & \frac{\partial}{\partial \tilde{z}}\tilde{U}_x \\ \frac{\partial}{\partial p}\tilde{U}_y & \frac{\partial}{\partial \tilde{u}_x}\tilde{U}_y & \frac{\partial}{\partial \tilde{u}_y}\tilde{U}_y & \frac{\partial}{\partial \tilde{z}}\tilde{U}_y \\ \frac{\partial}{\partial p}\tilde{Z} & \frac{\partial}{\partial \tilde{u}_x}\tilde{Z} & \frac{\partial}{\partial \tilde{u}_y}\tilde{Z} & \frac{\partial}{\partial \tilde{z}}\tilde{Z} \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta \tilde{u}_x \\ \Delta \tilde{u}_y \\ \Delta \tilde{z} \end{bmatrix} = -\begin{bmatrix} resC \\ res\tilde{U}_x \\ res\tilde{U}_y \\ res\tilde{Z} \end{bmatrix}$$

}

CVFEM Sample Code: Diffusion

```
// start the assembly
for ( int ip = 0; ip < AlgTraits::numScsIp_; ++ip ) {

  // left and right nodes for this ip
  const int il = lrscv_[2*ip];
  const int ir = lrscv_[2*ip+1];

  // compute ip property
  DoubleType diffFluxCoeffIp = 0.0;
  for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {
    const DoubleType r = v_shape_function_(ip,ic);
    diffFluxCoeffIp += r*v_diffFluxCoeff(ic);
  }

  // assemble to rhs and lhs
  DoubleType qDiff = 0.0;
  for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {
    DoubleType lhsfacDiff = 0.0;
    for ( int j = 0; j < AlgTraits::nDim_; ++j ) {
      lhsfacDiff += -diffFluxCoeffIp*v_dndx(ip,ic,j)*v_scs_areav(ip,j);
    }
    qDiff += lhsfacDiff*v_scalarQ(ic);

    // lhs; il then ir
    lhs(il,ic) += lhsfacDiff;
    lhs(ir,ic) -= lhsfacDiff;
  }

  // rhs; il then ir
  rhs[il] -= qDiff;
  rhs[ir] += qDiff;
 }
}
```

$$\int w \frac{\partial q_j}{\partial x_j} dV \approx -\sum_{ip} \frac{\mu}{Sc}_{ip} \frac{\partial \phi}{\partial x_j}_{ip} n_j dS = -\sum_{ip} \frac{\mu}{Sc}_{ip} \sum_{nd} \frac{\partial N^{ip}_{nd}}{\partial x_j} \phi_{nd} A^{ip}_j$$

CVFEM Sample Code: Time

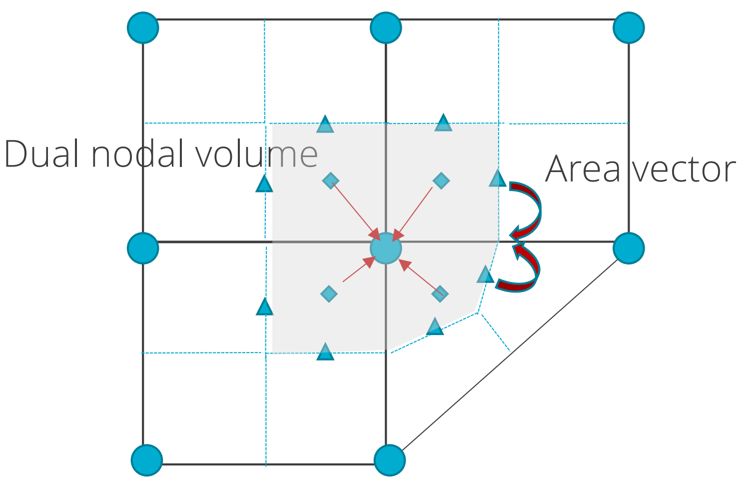
```
for ( int ip = 0; ip < AlgTraits::numScvIp_; ++ip ) {

   // nearest node to ip
   const int nearestNode = ipNodeMap_[ip];

   // zero out; scalar
   DoubleType qNm1Scv = 0.0;
   DoubleType qNScv = 0.0;
   DoubleType qNp1Scv = 0.0;
   DoubleType rhoNm1Scv = 0.0;
   DoubleType rhoNScv = 0.0;
   DoubleType rhoNp1Scv = 0.0;

   for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic}) {
     // save off shape function
     const DoubleType r = v_shape_function_(ip,ic);

     // scalar q
     qNm1Scv += r*v_qNm1(ic);
     qNScv += r*v_qN(ic);
     qNp1Scv += r*v_qNp1(ic);

     // density
     rhoNm1Scv += r*v_rhoNm1(ic);
     rhoNScv += r*v_rhoN(ic);
     rhoNp1Scv += r*v_rhoNp1(ic);
   }
```

```
   // assemble rhs
   const DoubleType scV = v_scv_volume(ip);
   rhs(nearestNode) +=
     -(gamma1_*rhoNp1Scv*qNp1Scv + gamma2_*rhoNScv*qNScv +
gamma3_*rhoNm1Scv*qNm1Scv)*scV/dt_;

   // manage LHS
   for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {
     // save off shape function
     const DoubleType r = v_shape_function_(ip,ic);
     const DoubleType lhsfac = r*gamma1_*rhoNp1Scv*scV/dt_;
     lhs(nearestNode,ic) += lhsfac;
   }
```

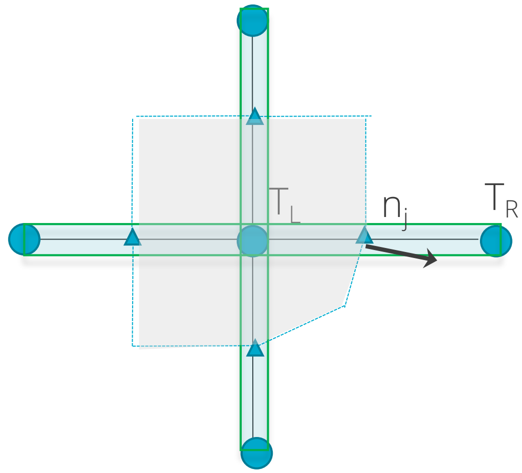
$$\int w \frac{\partial \rho \phi}{\partial t} dV \approx \sum_{ip} \frac{\left(\gamma_1 \rho_{ip}^{n+1} \phi_{ip}^{n+1} + \gamma_2 \rho_{ip}^{n} \phi_{ip}^{n} + \gamma_3 \rho_{ip}^{n-1} \phi_{ip}^{n-1}\right)}{\Delta t} V_{ip}$$

The Control Volume for EBVC is Defined by the ***Dual-Volume***

- All primitives are collocated at the vertices of the elements with equal-order interpolation
- A dual mesh is constructed to obtain flux and volume quadrature locations
- Classic two-state, “L” and “R” approach provides spatially second-order accuracy
- Iterate ***Nodes*** for volume-based contributions
- Iterate ***Edges*** for surface-based contributions



Dual-volume definition



Edge-based stencil

An EBVC Algorithm

Regardless of topology (restricted to low-order), edges are unique

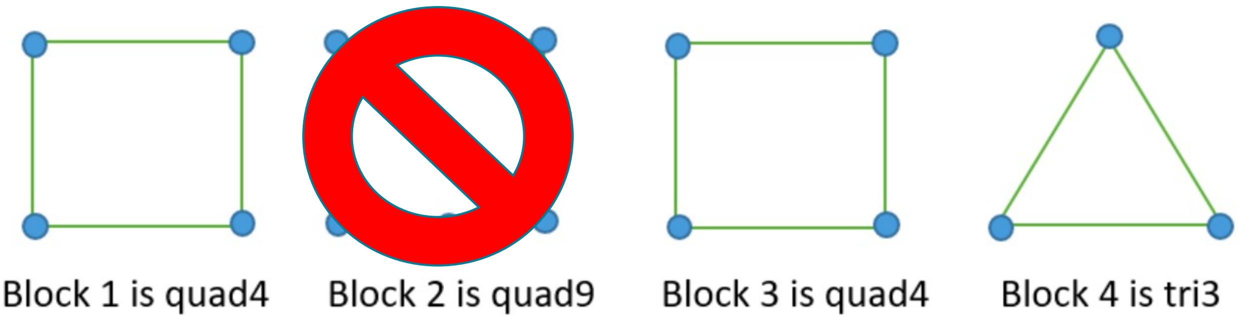
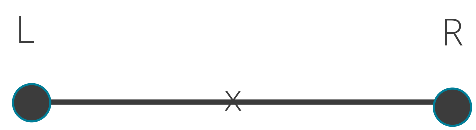


Figure 4: Heterogeneous topologies example.

Therefore, the topology over which we loop is the edge2 with “L” and “R” nodes, respectively, and the single nodal

EBVC Sample Code: Diffusion

```
for ( stk::mesh::Bucket::size_type k = 0 ; k < length ; ++k ) {

      // get edge
      stk::mesh::Entity edge = b[k];

      // left and right nodes
      stk::mesh::Entity nodeL = edge.edge_node_rels[0];
      stk::mesh::Entity nodeR = edge.edge_node_rels[1];

      const double viscIp = 0.5*(diffFluxCoeffL + diffFluxCoeffR);
           const double qNp1L = *stk::mesh::field_data( scalarQNp1, nodeL );
      const double qNp1R = *stk::mesh::field_data( scalarQNp1, nodeR );

      double lhsfac = -viscIp*asq*inv_axdx;
      double diffFlux = lhsfac*(qNp1R - qNp1L) + nonOrth;

      // first left
      p_lhs[0] = -lhsfac;
      p_lhs[1] = +lhsfac;
      p_rhs[0] = -diffFlux;

      // now right
      p_lhs[2] = +lhsfac;
      p_lhs[3] = -lhsfac;
      p_rhs[1] = diffFlux;
}
```



$$\frac{\partial \phi}{\partial x_j}\Big|ip = G_j^{ip}\phi + \left[(\phi_R - \phi_L) - G_l^{ip}\phi\Delta x_l\right]\frac{A_j^{ip}}{A_k \Delta x_k}$$

EBVC Sample Code: Time

```
void
ScalarMassBDF2NodeSuppAlg::node_execute(
  double *lhs,
  double *rhs,
  stk::mesh::Entity node)
{
  // deal with lumped mass matrix
  const double qNm1       = *stk::mesh::field_data(*scalarQNm1_, node);
  const double qN         = *stk::mesh::field_data(*scalarQN_, node);
  const double qNp1       = *stk::mesh::field_data(*scalarQNp1_, node);
  const double rhoNm1     = *stk::mesh::field_data(*densityNm1_, node);
  const double rhoN       = *stk::mesh::field_data(*densityN_, node);
  const double rhoNp1     = *stk::mesh::field_data(*densityNp1_, node);
  const double dualVolume = *stk::mesh::field_data(*dualNodalVolume_, node);
  const double lhsTime    = gamma1_*rhoNp1*dualVolume/dt_;
  rhs[0] -= (gamma1_*rhoNp1*qNp1 + gamma2_*qN*rhoN + gamma3_*qNm1*rhoNm1)*dualVolume/dt_;
  lhs[0] += lhsTime;
}
```

$$\int w \frac{\partial \rho \phi}{\partial t} dV \approx \sum_{nd} \frac{\left(\gamma_1 \rho_{nd}^{n+1} \phi_{nd}^{n+1} + \gamma_2 \rho_{nd}^{n} \phi_{nd}^{n} + \gamma_3 \rho_{nd}^{n-1} \phi_{nd}^{n-1}\right)}{\Delta t} V_{nd}$$

EBVC Sample Code: Complex LES-based Example

```
void
TurbKineticEnergyKsgsNodeSourceSuppAlg::node_execute(
  double *lhs,
  double *rhs,
  stk::mesh::Entity node)
{
  // filter
  double filter = std::pow(dualVolume, 1.0/nDim_);

  int nDim = nDim_;
  double Pk = 0.0;
  for ( int i = 0; i < nDim; ++i ) {
    const int offSet = nDim*i;
    for ( int j = 0; j < nDim; ++j ) {
      Pk += dudx[offSet+j]*(dudx[offSet+j] + dudx[nDim*j+i]);
    }
  }
  Pk *= tvisc;

  double Dk = cEps*rho*std::pow(tke, 1.5)/filter + lrksgsfac_*2.0*visc*dsqrtkSq;

  if ( Pk > tkeProdLimitRatio_*Dk )
    Pk = tkeProdLimitRatio_*Dk;

  rhs[0] += (Pk - Dk)*dualVolume;
  lhs[0] += 1.5*cEps*rho*std::sqrt(tke)/filter*dualVolume;
}
```

$$RHS = P_k - D_k$$

$$P_k = 2\mu^T \tilde{S}^*_{ij} \frac{\partial \tilde{u}_i}{\partial x_j} \quad D_k = \rho C_\epsilon \frac{k^{1/2}_{SGS}}{\Delta}$$

$$S^*_{ij} = \tilde{S}_{ij} - \frac{1}{3}\frac{\partial \tilde{u}_k}{\partial x_k}\delta_{ij}$$

CVFEM Sample Code: Complex LES-based Example (snippet)

```
for (int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic) {
    for (int ip=0; ip < AlgTraits::numScvIp_; ++ip) {

     const DoubleType r = v_shape_function_(ip, ic);
     tkeIp += r*v_tkeNp1(ic);
     rhoIp += r*v_densityNp1(ic);
     tviscIp += r*v_tvisc(ic);
     dualNodalVolIp += r*v_dualNodalVolume(ic);
     cEpsIp += r*v_cEps(ic);
     viscIp += r*v_visc(ic);
     for ( int i = 0; i < AlgTraits::nDim_; ++i ) {
       const DoubleType sqrtk = stk::math::sqrt(v_tkeNp1(ic));
       w_dsqrtkdx[i] += v_dndx(ip,ic,i)*sqrtk;
       const DoubleType ui = v_velocityNp1(ic,i);
       for ( int j = 0; j < AlgTraits::nDim_; ++j ) {
         w_dudx[i][j] += v_dndx(ip,ic,j)*ui;
       }
     }
   }
}
```

$$RHS = P_k - D_k$$

$$P_k = 2\mu^T \tilde{S}^*_{ij} \frac{\partial \tilde{u}_i}{\partial x_j} \qquad D_k = \rho C_\epsilon \frac{k^{1/2}_{SGS}}{\Delta}$$

$$S^*_{ij} = \tilde{S}_{ij} - \frac{1}{3}\frac{\partial \tilde{u}_k}{\partial x_k}\delta_{ij}$$